Vulnerability Assessment in Autonomic Networks and Services: A Survey

Martín Barrère, Rémi Badonnel, Olivier Festor

Vulnerability Assessment in Autonomic Networks and Services: A Survey

Martín Barrère, Rémi Badonnel and Olivier Festor
INRIA Nancy Grand Est - LORIA, France
Email: {barrere, badonnel, festor}@inria.fr

***Abstract*—Autonomic networks and services are exposed to a large variety of security risks. The vulnerability management process plays a crucial role for ensuring their safe configurations and preventing security attacks. We focus in this survey on the assessment of vulnerabilities in autonomic environments. In particular, we analyze current methods and techniques contributing to the discovery, the description and the detection of these vulnerabilities. We also point out important challenges that should be faced in order to fully integrate this process into the autonomic management plane.**

***Index Terms*—Vulnerability assessment, autonomic computing, computer security, vulnerability management.**

I. Introduction

The growing development of networks and the multiplication of the services offered over them have dramatically increased the complexity of network management. The paradigm of autonomic computing has been introduced for addressing this complexity through the design of networks and services which are responsible of their own management [1], [2]. While high-level objectives are provided by network administrators, management operations are delegated to the networks themselves thus alleviating the administrative burden required for maintaining large-scale expanding networks as well as dozens of heterogeneous services [3], [4]. Such approach aims at providing strong foundations for developing scalable and flexible infrastructures able to support a demanding and changing technological reality.

In that context, autonomic networks involve a set of functional areas called self-* properties that organize their self-governing nature. Four main areas classify the mechanisms used for regulating their behavior, namely, self-configuration providing means for automatically configuring components and services, self-optimization covering techniques for monitoring and adapting parameters in order to achieve an optimal operation according to the laws that govern the system, self-healing for automatically detecting, diagnosing and repairing localized software and hardware problems, and self-protection methods supporting activities for identifying and defending the system against potential threats [5]. Autonomic entities work under closed control loops that govern their behavior. A closed loop controls the dynamic behavior of the system by consuming not only external inputs from the environment but also its own output by means of a feedback mechanism. The sequence of phases determining the behavior of a specific autonomic entity includes monitoring its current state, analyzing the available information, planning future actions, and executing generated plans compliant to specified high-level goals [6]. Self-* properties are intended to autonomously solve high-level requirements, however, their implementation is complex and poses hard challenges. Along with administration tasks done by humans, changes performed by autonomic entities may inadvertently generate vulnerable states when following high-level objectives. Even though these changes can operationally improve the environment, insecure configurations may be produced increasing the exposure to security threats. Thus, enabling autonomic networks and systems to manage vulnerabilities and maintain safe configurations constitutes a major challenge.

In computer security, a vulnerability can be understood as a flaw or weakness in system security procedures, design, implementation, or internal controls that could be exercised (accidentally triggered or intentionally exploited) and result in a security breach or a violation of the system's security policy [7], [8]. Under this perspective, vulnerability management is a cross-cutting concern strongly related but not limited to self-configuration and self-protection activities of autonomic networks. This process is depicted in Fig. 1 where a control loop enables the assessment and remediation of potential vulnerable states generated by both administrators tasks and self-management activities, thus securing the environment. The aim of this survey is to investigate methods and techniques contributing to vulnerability management in such autonomic

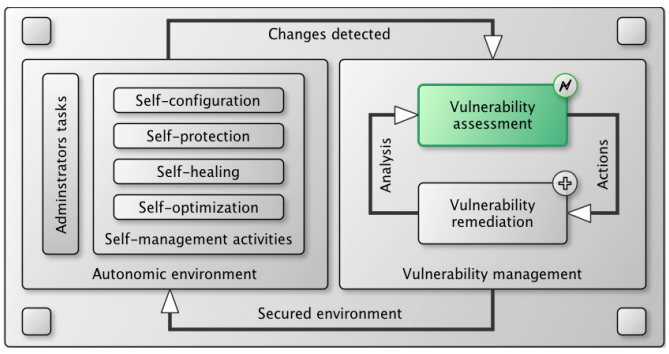

Fig. 1: Positioning of vulnerability management with respect to self-management activities

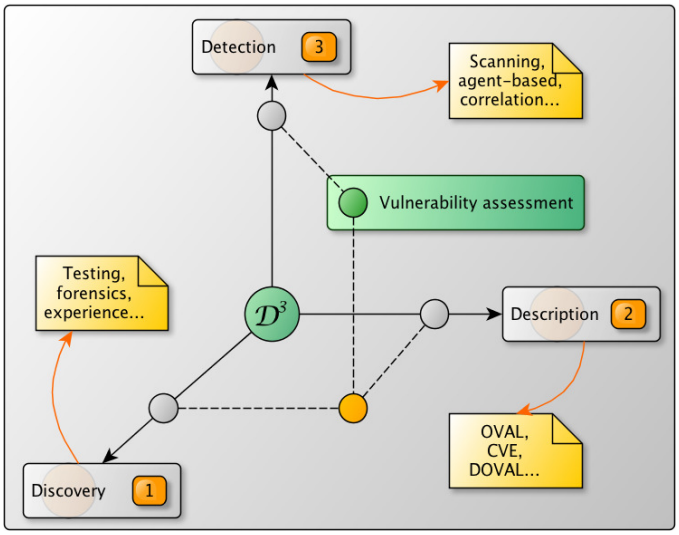

Fig. 2: Vulnerability assessment - D^3 classification

environments, specially focused on the vulnerability assessment process. We decompose vulnerability assessment activities by considering what we call a D^3 (*D cube*) classification as illustrated in Fig. 2, D^3 standing for *Discovery, Description* and *Detection*.

The D^3 classification provides a basis, divided into three axes, for organizing the foundations of vulnerability assessment which is the heart of this survey and that in turn constitutes the first step for the vulnerability management process to be embedded into autonomic environments. Autonomic entities must be provided with knowledge about current vulnerabilities, either with mechanisms for discovering threats by themselves or with machine-readable specifications about security alerts. Regardless of the mechanism chosen, vulnerability discovery techniques (axis 1) must be analyzed in order to unveil unknown vulnerabilities, and to explore and understand the constantly evolving threatening environment. Taking advantage of these mechanisms, new knowledge becomes available for increasing the vulnerability awareness of self-governed environments. Such consciousness must be formally specified in order to be understood by computing devices, thus standard languages and protocols must be provided for describing and exchanging security advisories (axis 2). Such security knowledge increases the capability of autonomic networks and systems for detecting vulnerabilities in the surrounding environment (axis 3) and provides a strong support for taking decisions when performing self-management activities.

The remainder of this survey is structured as follows. Section II introduces fundamentals of vulnerability management and presents its positioning with respect to change management and risk assessment, connecting these concepts with autonomic networks and systems requirements. Section III presents the approaches used for vulnerabilities discovery. Section IV discusses current mechanisms and protocols for describing vulnerabilities, analyzing computer systems and exchanging security related content. Section V presents techniques for assessing device vulnerabilities as well as network vulnerabilities, and the correlation with security threats and attack graphs. Section VI identifies major challenges and proposes directions for further work. Section VII closes this survey presenting the obtained conclusions.

II. Vulnerability Management

Managing large-scale networks is a complex task and by nature, humans make errors when configuring them. In addition, changes performed by autonomic entities may increase their own security exposure. Because of this, vulnerable configurations are likely within such environments and they may potentially lead to a wide spectrum of negative and unwanted issues such as instability, unavailability, confidentiality problems, and many more. Usually, the risk level of a system is based on three main combined factors, namely, the potentiality of a threat in conjunction with the exposure of that system to such threat and the impact that a successful attack related to this threat may have in that system [9]. The exposure of a system in turn is directly related to the vulnerabilities present in such system, thus managing vulnerabilities that might be exercised by a given threat constitutes a critical activity for quantifying the system exposure and hence the risk level of autonomic networks and systems. This survey is targeted on vulnerability assessment, an essential part of the vulnerability management process. However, it is important to have a clear image of the domain where vulnerability assessment activities take place before entering into deeper details. This section is intended to explain the general process of dealing with vulnerabilities in order to understand the interconnection between the involved management activities as well as to identify direct and orthogonal works done in the field that may potentially contribute to such process.

Vulnerability management is usually defined as the practice of (I) identifying, (II) classifying, (III) remediating and mitigating vulnerabilities [10]. Historically, the process of managing vulnerabilities has been exercised for long time in different fields. Military for instance considers a vulnerability as the inability to withstand an adverse circumstance produced by a hostile environment. Therefore, security procedures are defined to state how to proceed in these situations, thus constituting part of a vulnerability management program [11]. In information technology, vulnerabilities have existed from the beginning. As an example, in 1903 the Marconi wireless telegraph was reported to contain a flaw that allowed an attacker to intercept any message sent by the device thus leading to unauthorized information disclosure [12]. In 1962, the Multics CTSS operating system running on IBM 7094 was reported to have a flaw allowing an unauthorized user to disclose the password of every user on the system. Reports of identified vulnerabilities have continued coming up since the 60's until our days. With the incursion of computing systems into human activities, the diversification of programs and services have set up more and more vulnerabilities compromising the security of such systems. These undesired effects made it clear the need of developing security programs able to deal with such security issues. In 1972, a computer

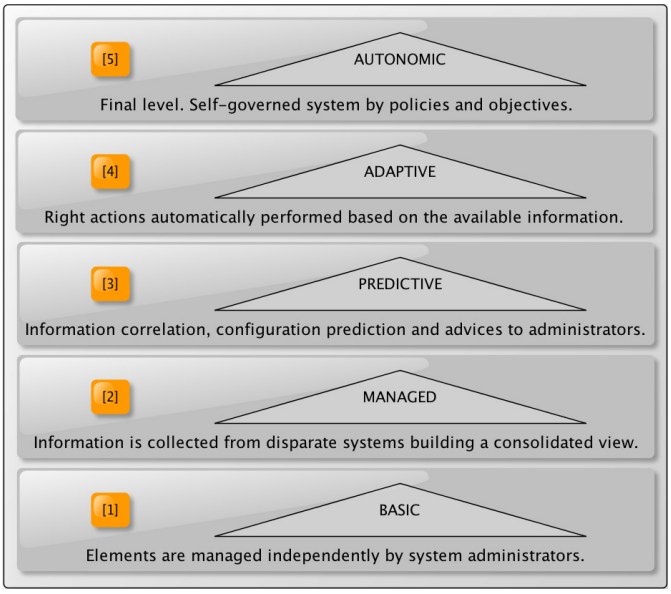

Fig. 3: Stages towards autonomic computing [15]

security technology planning study was created by the U.S. Air Force Systems Command (AFSC). The objective of such program was to specify directives for securing the use and development of computing systems [7]. Since those days, managing vulnerabilities became an essential activity for any organization involving the use of computers or telecommunications equipment. Nowadays, several technologies that will be later detailed in this survey are widely used for supporting this process such as the Common Vulnerabilities and Exposures system (CVE) [13] for enumerating known vulnerabilities or the Security Content Automation Protocol (SCAP) [14] for automating vulnerability management activities.

In order to integrate vulnerability management mechanisms into autonomic environments, it is important to consider what levels of automation can be achieved. As proposed in [15], the evolutionary path to autonomic computing is represented by five levels as illustrated in Fig. 3. The first level depicts the basic approach where network elements are independently managed by system administrators. The management of network elements by collecting information from disparate systems into one consolidated view constitutes the managed level. At the predictive level, new technologies are incorporated for correlating information, predicting optimal configurations and providing advices to system administrators. The ability to automatically take actions based on the available information constitutes the adaptive level. Finally, the autonomic level is achieved when the system is governed by business policies and objectives. These objectives define the purpose of the autonomic system that will be inherently pursuit by its logical implementation and supported by its internal know-how as well as its ability to sense the environment. It is important to highlight that a wide range of software and systems could embody autonomic solutions to some extent if they can be adapted to interact with the environment by means of sensors and effectors and to work guided by rules and policies intended to achieve a specific purpose.

Under this perspective, the establishment of a secure process for dealing with vulnerabilities requires the specification of a policy defining the desired system state and a well-known secure initial state to identify vulnerabilities and policy compliance [16]. The main activities performed during the lifecycle of the vulnerability management process can be mapped to the same activity line present in autonomic components. Fig. 4 describes the general lifecycle of an autonomic component where the main activities done for dealing with vulnerabilities have been mapped to the task loop performed during the autonomic manager lifecycle. The resource manager interface provides means for monitoring and controlling the managed element (hardware, software, others). While its sensor enables the autonomic manager to obtain data from the resource, the effector allows it to perform operations on the resource. The autonomic manager is composed of a cycle of four activities and also exposes a manageability interface, in the same way the managed element does. This interface allows other autonomic managers to use its services and provides composition capabilities on a distributed infrastructure. Based on the specified directives, the autonomic manager will continuously monitor the managed element and will perform an analysis of the perceived state. As shown in Fig. 4, vulnerability identification activities take place in this monitoring phase where tasks for assessing and analyzing vulnerable states are performed (I) taking advantage of the available security knowledge. When a security problem is found, it is classified (II) and changes for correcting the situation must be performed. Therefore, vulnerability counter-measures are planned based on several factors such as importance, risks and impact. Finally, a change plan is generated and remediation tasks are executed (III) in order to maintain safe configurations and to be compliant with the current policy.

As we mentioned before, computer systems management is becoming more and more complex over time. Conventional approaches are not scalable, thus leading to new management problems. Autonomic systems release administrators from low-level details. The self-management approach works on a high-level, goal-oriented basis. This scenario allows to specify how things work while functional details are solved by the underlying autonomic system. However, when vulnerability management related tasks are performed, several changes are introduced in the working environment as well. Thus, it is important to count on mechanisms and techniques for assessing and evaluating the impact and the effectiveness of these changes.

Change management already constitutes a challenging activity when performed by human administrators, the automation of such process is even more complex. As a simple example, we can imagine a system that has installed a specific software X with version 1.0 providing services A and B. This version has a vulnerability that if exploited would allow an attacker to perform an unauthorized access to the system. In light of this, it is decided to upgrade the software X to its version 1.1 where the referred vulnerability has been eradicated. However, it turns out that the new version does not provide service B any more but it is still required by software Y. Different actions could be taken in order to face this problem, e.g.,

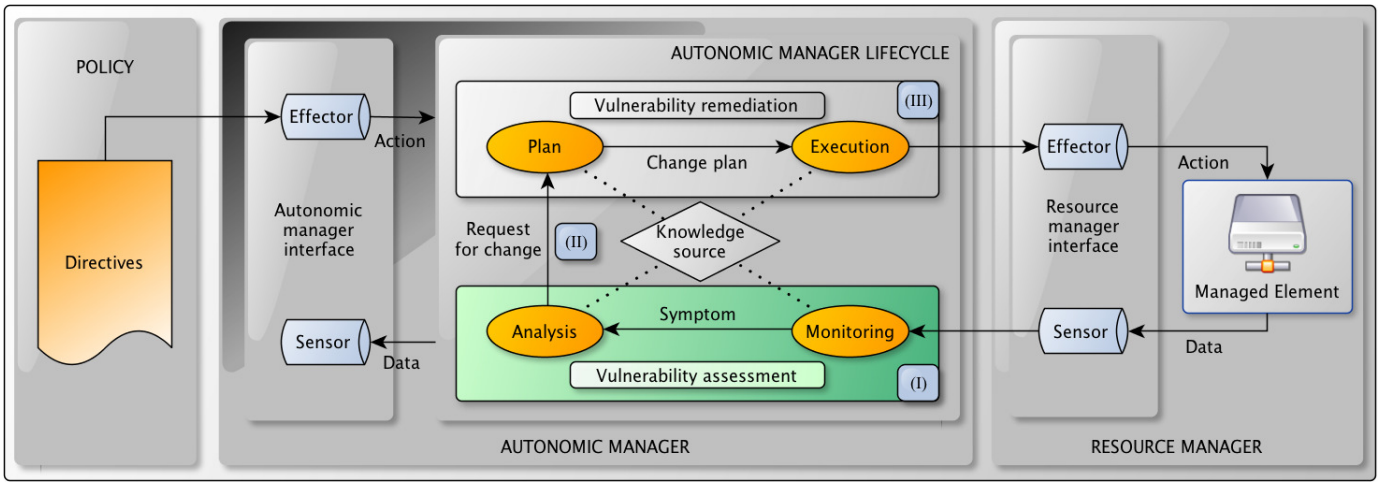

Fig. 4: Mapping of the vulnerability management activity into the autonomic lifecycle [6]

removing software Y if it is not required, providing service B with other not vulnerable piece of software if possible, keeping X in its vulnerable version 1.0, and so on. The point is that this decision is based on factors that define the nature of the system, laws that rule the behavior and purpose of the system. While some entities will prioritize functionality over security, others might follow the other way around. But most importantly, no matter what the chosen action is, performed changes should be effective as to the objective they were designed for and consistent with the rules that govern the system. The latter is not always easy to achieve thus mechanisms for solving conflicts and techniques for reducing the impact of these changes must be taken into account as well. A few works really address vulnerability management in autonomic networks and systems. Orthogonal works have been proposed in the area of change management. They contribute to ensure the correctness of configuration operations and their positive impact over services but they do not consider security aspects with respect to vulnerable configurations. Therefore, vulnerability management activities and change management techniques become interconnected; network changes must be evaluated in order to ensure safe modifications and at the same time, vulnerable states must be remediated by performing controlled changes in the environment. Within this section, we point out related work about change management on networks and systems considering vulnerability treatments, and we also cover different approaches contributing to the risk assessment activity.

A large variety of techniques have been proposed to evaluate the impact of changes in networks and systems. These contributions provide strong foundations for their automation. Information Technology Service Management (ITSM) is a fundamental work field for institutions and corporations; intended to expose mechanisms for dealing with changes within an organization, trying to minimize the impact and, at the same time, maximize the utility provided by them [17]. Within the vulnerability management scenario, vulnerability mitigation and remediation involve changes on the underlying infrastructure, thus requiring impact analysis. It is well known that IT processes evolve over time. Several reasons support this evolution; one of them is the existence of optimization points. An improved process may decrease its complexity and therefore becoming more understandable and less error prone, leaving smaller space for security problems. The work proposed in [18] for instance introduces a model for relating IT management complexity metrics to key business-level performance metrics like time and labor cost. The approach allows to determine if a given process transformation is likely to improve business performance (e.g. in terms of time) based on process associated complexity metrics. Some approaches consider future changes already at system design, thus changes are more manageable when they are required. Design rationale is one of these approaches, which involves an explicit documentation of the reasons behind the decisions taken, providing a better view of system dynamics. An approach based on design rationale has been proposed in [19] where the method is used to represent the causal relationships between architecture design elements and decisions. Based on Bayesian belief networks, the model is capable of capturing the probabilistic causal relationships between design elements and decisions, and analyzing design change impact.

Assessing change associated risks also provides a key support for change management. When a change is needed on an IT infrastructure, assessing techniques are commonly used to evaluate or estimate the projected change associated risks in order to take decisions about its effective implementation. Sometimes, these decisions are not one hundred percent clear, therefore, measurable mechanisms are required in order to perform appropriate cost-benefit analysis. The work presented in [20] describes a method for evaluating the risk exposure associated with a change. Based on the obtained risk exposure metric, the authors present an automatic mechanism for assigning priorities to changes and therefore, allowing to organize and take business-level decisions about required changes. Previous experience is normally used when facing new problems, trying to find adaptability points and reuse successful actions taken in the past. A solution for automating the risk assessment process using past experience has been proposed in [21]

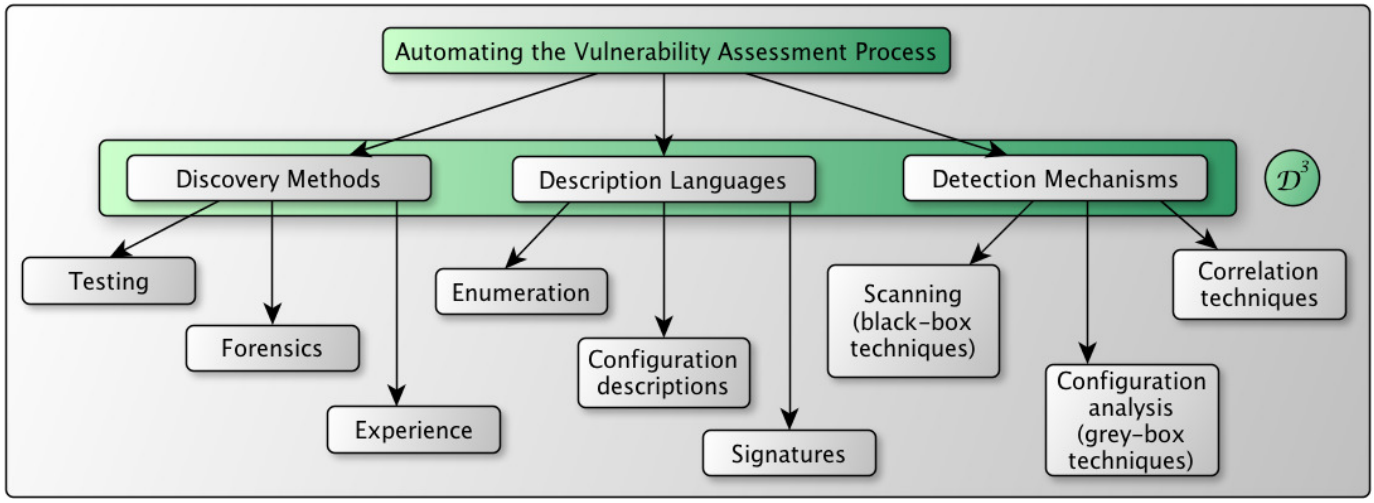

Fig. 5: Automated vulnerability assessment classification

where historical data –from previous sequences of changes– is combined in order to analyze the impact of changes over the affected elements. The impact severity on these elements is a crucial piece of information for the organization when taking changes decisions. The work presented in [22] proposes a model for analyzing the business impact of operational risks resulting from change related service downtimes of uncertain duration, considering the existing interdependence between services. Analysis results are used later for scheduling changes with the lowest expected impact on the business. Sometimes, some fundamental business services are so important –this is, with a high negative business impact if the change fails– that organizations often opt finding different and more secure approaches to implement the required change. From an autonomic point of view, automated techniques to assess change associated risks like those presented here, are extremely important in order to achieve these goals.

It is also important to prevent states which are vulnerable from a security perspective and not only from an operational viewpoint. Increasing systems vulnerability awareness is an important challenge within autonomous behavior as it provides means for better understanding the surrounding environment. As the autonomic nervous system, autonomic systems and networks must be able to perform diagnosis on the environment they are working on. Because systems high-level state is typically changing, these capabilities provide the basis for adaptation. Awareness of risks and vulnerabilities is one of them and provides the first step towards more secure systems. A methodology for minimizing the dissemination of vulnerabilities on hybrid networks has been proposed in [23]. Within such multi-agent system, the main task of each software agent is to detect vulnerabilities and exposures. Each agent can exchange security notifications with other agents in order to warn them about possible threats or network attacks. Combinations of powerful techniques, as we will further present, provide a strong basis for vulnerability detection. Nevertheless, in practice, it is almost impossible to be aware of each security and exploitable hole for each system. Vulnerability detection provides large amounts of information that allow systems to be aware of threats, but autonomic systems need to see the big picture, not only as a snapshot but also considering past experience, in order to identify risk factors and perform progressive adaptation to achieve secure states. The work presented in [24] and [25] proposes a security metric-based framework that identifies and quantifies objectively security risk factors, including existing vulnerabilities, historical trend of vulnerabilities of remotely accessible services, prediction of potential vulnerabilities for any general network service and their estimated severity and finally propagation of an attack within the network.

As stated before, the main target of this survey is to deeply explore existing mechanisms and techniques that contribute to the vulnerability assessment process. This activity increases the vulnerability awareness of autonomic environments that along with the remediation activity completes the vulnerability management process loop. In order to cover the automation of the vulnerability assessment process, we propose the classification depicted in Fig. 5 where we divide the activity into three main areas following the D^3 approach. First, we will present current approaches for discovering unknown vulnerabilities connecting their applicability over autonomic environments. Then, we will detail description languages capable of representing security advisories about known vulnerabilities. Finally, we will describe techniques that take advantage of such knowledge for performing security analysis from both, agent and network perspectives, in a centralized or a distributed manner.

III. Discovering Vulnerabilities

In the previous section we reviewed some important aspects of vulnerability management in autonomic networks and discussed approaches to face related infrastructure changes. In order to perform these security improvement changes, the ability to unveil threats present on the environment becomes an essential requirement. Because the whole set of potential vulnerabilities on each system is typically unknown, techniques for learning and discovering vulnerabilities must be developed. Under this perspective, it is also important to consider how

such security information actually becomes available for protecting autonomic networks and systems. Indeed, there is a bigger ecosystem that not only involves vulnerabilities and security defects but also people and their motivations [26]. New vulnerability information usually follows complex paths before users get benefited of it [27]. This security ecosystem is frequently governed by economical laws where buyers and sellers of new security findings establish complex vulnerability markets. It is not the target of this survey to cover how vulnerability information is traded in both white and black markets, however it is important to keep in mind that computer security is not only about technologies but also about people's behavior and motivations. In this section we will focus on mechanisms and means for discovering unknown vulnerabilities. This topic has been barely addressed within the field of autonomic networks, therefore the objective is to explore different approaches that can be potentially integrated into self-governed systems. Usually, almost every solution designed for standard systems can be embedded to some extent into autonomic closed loops. The ease of such integration depends on the nature of these approaches, however, we can normally think or design autonomic elements with sensors capable of consuming the required input that will feed the existing solution and adapting the performed actions to be wrapped by autonomic effectors. While automating the operation of these solutions might be in some cases quite straightforward, making them self-adaptive to the surrounding environment as well as to work under a policy-based perspective (autonomous) constitutes an open and highly challenging problem. In light of this, a subset of prominent perspectives has been selected to provide an overview of available strategies for unveiling security issues. Our research includes some of the most studied fields including testing methods, network forensics techniques and case-based reasoning.

A. *Exploiting testing methods*

Exploiting testing methods provides a powerful approach to unknown vulnerability detection. Software applications are commonly designed with a set of specific goals in mind in order to provide effective solutions to the stated requirements. While developers pursue efficient functional constructions, testers perform tasks for identifying correctness, completeness, quality and security of developed computer software. Several approaches under the tester point of view are used when software tests are designed [28]. White-box testing allows testers to have access to internal structures, algorithms and the code itself of the software being tested, e.g. static analysis, code review. Black-box testing on the other hand does not provide information about the internal implementation and the software is seen as an input-output black box, e.g. dynamic analysis, performance tests. Grey-box testing combines the previous approaches by considering knowledge about the software internals but executes the tests at a black-box level, e.g. internal database testing. Under a self-governing perspective, autonomic elements could be analyzed using these techniques in order to identify abnormal behavior. This first step would provide useful information to the underlying government mechanism about unsafe components. Even though traditional testing techniques unveil an important amount of software problems, it is unfortunately common for testers to focus on functionality correctness and to omit strong security tests. At the time of software construction and testing, normal input tests are frequently more numerous than anomalous input tests. Because of this, several unknown vulnerabilities remain hidden behind untested input data. Fuzzing techniques are intended to find these kind of software bugs.

The fuzzing approach complements traditional testing to discover combinations of code and data by exploiting the power of randomness, protocol knowledge, and attack heuristics. Instead of using normal input data, fuzzing methods generate unexpected or malformed inputs for feeding the target software. Software behavior is then assessed in order to identify potential vulnerability hotspots. A wide view of current fuzzing techniques is presented in [29] where different approaches are explored, highlighting fuzzing contributions to vulnerability detection. Since application's input space is in most cases impossible to enumerate, fuzzing techniques use two main approaches: data generation and data mutation (randomly modifying well-formed inputs). However, traditional fuzzing tools present some randomness related drawbacks when working with applications that perform various types of integrity checks (e.g. checksum). Checksum mechanisms reject an important part of the generated input set at initial execution stages, decreasing the fuzzer effectiveness and code coverage. The work presented in [30] identifies the stated problems and presents an approach to overcome early malformed input rejection due to checksum failures. Other approaches have been proposed as well for avoiding input related issues. A fuzzing-based methodology called configuration fuzzing has been presented in [31] where the configuration of the running application is randomly modified at certain execution points in order to check for vulnerabilities that only arise in certain conditions. The proposed approach is performed within the running environment over a copy of the original application, enabling the detection of missed vulnerabilities before the application release. Considering the fact that autonomic systems are ruled by high-level policies, the same mechanism could be used for specifying properties and the expected behavior of a piece of software. This would allow testing solutions to be embedded into self-governing entities in order to analyze their operation by checking the current state against the defined policies. As a first step this process could inform administrators about abnormal or unexpected behavior. However, it also could be taken one step further by automatically generating reports about the current system configuration for future use and looking for available solutions, configuration changes and patches.

B. *Using network forensics*

Techniques based on network forensics can also be used for discovering unknown vulnerabilities. Using network forensics confers useful methods that can highly contribute to this activity. Network forensics is typically known as the process of archiving all network traffic and analyzing subsets as

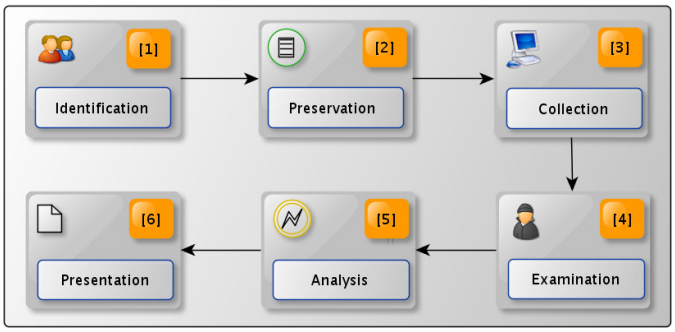

Fig. 6: Forensic investigation process [33]

necessary [32]. This activity generally involves traffic reconstruction to assess network activity, providing useful information for further network-related events analysis. Network forensics belongs to a wider computing field called digital forensics. Digital forensics is defined as the use of scientifically derived and proven methods towards the preservation, collection, validation, identification, analysis, interpretation, documentation and presentation of digital evidence derived from digital sources for the purpose of facilitating or furthering the reconstruction of events found to be criminal, or helping to anticipate unauthorized actions shown to be disruptive to planned operations [33]. Fig. 6 describes the stages involved within a forensic investigation.

Digital forensics benefits go beyond criminal prosecution. Several applications arise from the forensic discipline including powerful techniques usable in the vulnerability management area. Numerous contributions to computer security have been born within this field and they are widely used over different scenarios [32], [34]. Even though they are mostly targeted on traditional networked environments and not self-governing systems, these works provide a strong basis for being integrated into autonomic networks. Digital forensics provides a deep understanding of discovering mechanisms about the anatomy of an attack, how to gather pieces of evidence and put them together in order to determine how an attack took place on the system, when it was committed, who are the perpetrators and where they come from. Because of this, its robust technical background on data collection and analysis establishes a solid framework for performing computer system investigations, thus providing support to vulnerability management activities. The work presented in [34] provides an overview of digital forensics methodologies, computer and network vulnerabilities and security measures, and forensics tracking mechanisms to detect and deter intruders.

Forensic tools are extremely important on forensic scenarios. Effectiveness, efficiency, reproducibility, evidence consistency and integrity, traceability, security, are some of several factors that have to be considered when designing a forensic tool. Depending on the type of environment where the tool will be used, the previous features are required for a successful activity execution. Fundamental concepts related to network forensics and important features that a network forensic analysis tool (NFAT) should implement are presented in [32], namely, NFAT place and purpose, data capture, traffic analysis, and NFAT interaction and discovery services it should provide. Such concepts also support decentralized approaches where various forensic tools can collaborate in order to analyze the whole network. Evidence collection is an essential stage within a digital forensic investigation. Results of this stage feed the analysis stage. Because of this, evidence collection on computing systems is a highly active study field. The work proposed in [35] presents a graph-based approach towards network forensics analysis. The key construction of such approach is the evidence graph model that facilitates evidence presentation and automated reasoning. Based on the evidence graph, a hierarchical reasoning framework is proposed that consists of two levels. Local reasoning aims at inferring the functional states of network entities from local observations, while global reasoning aims at identifying important entities from the graph structure and extract groups of densely correlated participants in the attack scenario. Such approaches may highly contribute to the integration and positioning of forensic actuators within autonomic environments aiming at identifying, analyzing and providing reports about suspicious or abnormal behavior, and therefore highly contributing to the first dimension of the D^3 approach.

C. Taking advantage of experience

Past experience in dealing with vulnerabilities strengthens the ability to face new security problems. Under this perspective, performing case-based reasoning also provide interesting and useful outlooks for detecting unknown vulnerabilities. Case-based reasoning (CBR) is a problem solving methodology which exploits past experience. Past experience is maintained in the form of problem-solution pairs, also called cases. On the arrival of new problems, solutions of similar past problems are used after appropriate adaptation. The work presented on [36] applies the CBR approach for enabling self-configuration capabilities in autonomic systems. The authors propose a model for restricting the case-base size, providing efficiency with no accuracy loss. This approach can be applied on unknown situations, where some kind of nearness concept may be used in order to classify how similar the new problem is to the problems already known. Using different algorithms, solutions for known similar problems can be modified to achieve the solution of the new problem. Indeed, an approach for dealing with fault management issues using CBR has been proposed in [37]. The authors outline a distributed case-based reasoning system over a self-organizing platform capable of assisting operators in finding solutions for faults. Such approaches can provide strong support for developing autonomic solutions based on previous experience. In addition, such previous experience can be thought as part of the know-how that autonomic systems use to operate themselves in order to achieve their purpose. Moreover, considering self-configuration as a response for covering and repairing vulnerable configurations, case-based reasoning strategies can provide expertise and feed a database of known vulnerabilities, which is the heart of the next sections.

System improvements usually provide new or better technological capabilities, however, they also carry new space for

security concerns as well. Discovering unknown vulnerability constitutes an important security feature of self-governing systems. A wide spectrum of methods and techniques may be used for achieving this point. This section has covered some of the most important and promising areas involving several approaches for evaluating networks and systems looking for software flaws and configuration misuse, from fuzzing methods (proactive) to forensics techniques (reactive). Even though there exist autonomic approaches for unveiling vulnerable configurations, our research work indicates that most of the prominent contributions are not oriented to self-governed environments. Taking advantage of such approaches remains as a challenging activity. Autonomic environments should incorporate these capabilities in order to become adaptive with the changing environment being able to unveil potential unknown security threats. In addition, we consider that no matter what technique is used for discovering vulnerabilities, describing vulnerabilities in a standardized and machine-readable manner is essential for integrating such approaches into the autonomic management plane. This topic constitutes the central point of the next section.

IV. Describing Vulnerabilities

By the time a vulnerability is discovered, a time span will occur before system administrators are noticed about its existence. Another time will pass before a corrective solution exists and yet another will pass until all systems are patched. Attacker's activity usually takes place during this period of time, that can last from a few hours to several months or years. Because of this, it is important to develop a robust background as well as mechanisms and techniques in order to establish consistent and uniform means for describing vulnerabilities, analyzing and detecting them, and exchanging related information. The Common Vulnerabilities and Exposures or CVE system [13] has been introduced by the MITRE Corporation [38] as an effort for standardizing the enumeration of known information security vulnerabilities. The CVE dictionary, widely used today, allows the community to be aware of current existing threats and exposures by providing unique identifiers to each known security alert as well as descriptions written in natural language. This is extremely useful for increasing the security awareness of autonomic systems. However, the CVE standard only provides means for informing about their existence but not for their assessment. Describing the anatomy of known vulnerabilities and the techniques developed with such purpose are fundamental as they provide essential means for dealing with vulnerability management. This knowledge can highly increase the know-how of self-governing systems providing strong support for developing and integrating autonomic security solutions.

During the last years, several approaches on vulnerability analysis have been taken. Vulnerability signatures have been widely used by intrusion prevention systems (IPS) and intrusion detection systems (IDS). They are intended to describe the characteristics of the input that lead the execution of a specific program to a vulnerable point and the state that such program must hold for the vulnerability to be exploited [39]. Vulnerability signatures are mostly used for analyzing traffic looking for specific patterns and detecting potential attacks. The work proposed in [39] contributes to the second dimension of the D^3 approach by automatically generating high coverage vulnerability-based signatures. However, there are no fully developed up-to-date standards available for their representation and the generation as well as their coverage still remains an open problem. In addition to this issue, IDS also lack of fully mature standards for exchanging alerts. The Intrusion Detection Message Exchange Format (IDMEF) is a data model to represent information exported by intrusion detection systems proposed by the Internet Engineering Task Force (IETF) but its status is currently experimental [40]. Much of the work done in vulnerability analysis has defined the assessment infrastructure using its own vulnerability specification language arising compatibility and interoperability problems. Languages such as VulnXML [41] and the Application Vulnerability Description Language (ADVL) [42] have been developed as an attempt to mitigate these problems and to promote the exchange of security information among applications and security entities. However, these languages are only focused on web applications covering a subset of the existing vulnerabilities in current computer systems.

In order to cope with the problems described previously, the Open Vulnerability and Assessment Language (OVAL) [43] supported by MITRE Corporation standardizes how to assess and report upon the machine state of computer systems. OVAL is an XML-based language thus it inherits all XML features like platform independence, interoperability, transportability and readability. The OVAL specification is supported by XML schemas which serve as both the framework and vocabulary for the language. These schemas specify what content is valid within an OVAL document and what is not. OVAL is organized in three main XML Schemas, namely, (i) the OVAL Definition Schema that expresses a specific machine state; (ii) the OVAL Characteristics Schema that stores configuration information gathered from a system; and (iii) the OVAL Results Schema that presents the output from a comparison of an OVAL Definition against an OVAL System Characteristics instance. Valid XML instances typically represent specific machine states such as vulnerable states, configuration settings and patch states. Usually, a vulnerability is considered as a logical combination of conditions that if observed on a target system, the security problem described by such vulnerability is present on that system. The OVAL language follows the same idea by considering a vulnerability description as an OVAL definition. An OVAL definition specifies a criteria that logically combines a set of OVAL tests. Each OVAL test in turn represents the process by which a specific condition or property is assessed on the target system. Each OVAL test examines an OVAL object looking for a specific OVAL state. Components found in the system matching the OVAL object description are called OVAL items. These items are compared against the specified OVAL state in order to build the OVAL test result. The overall result for the criteria specified in the OVAL definition will be built using the results of each referenced OVAL test.

We now put forward an illustrative OVAL example of a vulnerability description for the Cisco Internetwork Operating

```
1. <?xml version="1.0" encoding="UTF-8"?>
2. <oval_definitions xsi:schemaLocation="http://oval.mitre.org/XMLSchema/oval-
       definitions-5 oval-definitions-schema.xsd http://oval.mitre.org/XMLSchema
       /oval-definitions-5#ios ios-definitions-schema.xsd http://oval.mitre.org/
       XMLSchema/oval-common-5 oval-common-schema.xsd" xmlns="http://oval.mitre.
       org/XMLSchema/oval-definitions-5" xmlns:xsi="http://www.w3.org/2001/
       XMLSchema-instance" xmlns:oval="http://oval.mitre.org/XMLSchema/oval-
       common-5" xmlns:oval-def="http://oval.mitre.org/XMLSchema/oval-
       definitions-5">
3.   <generator>
4.     <oval:product_name>The OVAL Repository</oval:product_name>
5.     <oval:schema_version>5.7</oval:schema_version>
6.     <oval:timestamp>2010-06-18T15:02:46.614-04:00</oval:timestamp>
7.   </generator>
8.   <definitions>
9.     <definition id="oval:org.mitre.oval:def:6086" version="2" class="
       vulnerability">
10.      <metadata>
11.        <title>Cisco IOS SIP Denial of Service Vulnerability</title>
12.        <affected family="ios">
13.          <platform>Cisco IOS</platform>
14.        </affected>
15.        <reference source="CVE" ref_id="CVE-2008-3800" ref_url="http://cve.
       mitre.org/cgi-bin/cvename.cgi?name=CVE-2008-3800" />
16.        <description> Vulnerable SIP implementation ... </description>
17.        <oval_repository>
18.          <dates> .... </dates>
19.          <status> .... </status>
20.        </oval_repository>
21.      </metadata>

22.      <criteria operator="AND">
23.        <criterion comment="IOS vulnerable versions" test_ref="oval:org.mitre
       .oval:tst:9025" />
24.        <criterion comment="SIP Test using runnning config. result contains:
       5060" test_ref="oval:org.mitre.oval:tst:24211" />
25.      </criteria>
26.    </definition>
27.  </definitions>
```

Listing 1: Cisco IOS vulnerability example (part 1)

```
28.  <tests>
29.    <version55_test id="oval:org.mitre.oval:tst:9025" version="1" comment="
       IOS vulnerable versions" check_existence="at_least_one_exists" check="at
       least one" xmlns="http://oval.mitre.org/XMLSchema/oval-definitions-5#ios">
30.      <object object_ref="oval:org.mitre.oval:obj:6804" />
31.      <state state_ref="oval:org.mitre.oval:ste:4432" />
32.    </version55_test>
33.    <line_test id="oval:org.mitre.oval:tst:24211" version="1" comment="SIP
       Test using ip socket. config contains: 5060 .may generate few false
       positive" check_existence="at_least_one_exists" check="at least one" xmlns
       ="http://oval.mitre.org/XMLSchema/oval-definitions-5#ios">
34.      <object object_ref="oval:org.mitre.oval:obj:6385" />
35.      <state state_ref="oval:org.mitre.oval:ste:6946" />
36.    </line_test>
37.  </tests>

38.  <objects>
39.    <version55_object id="oval:org.mitre.oval:obj:6804" version="1" xmlns="
       http://oval.mitre.org/XMLSchema/oval-definitions-5#ios" />
40.    <line_object id="oval:org.mitre.oval:obj:6385" version="1" xmlns="http://
       oval.mitre.org/XMLSchema/oval-definitions-5#ios">
41.      <show_subcommand>show running-config</show_subcommand>
42.    </line_object>
43.  </objects>

44.  <states>
45.    <version55_state id="oval:org.mitre.oval:ste:4432" version="1" xmlns="
       http://oval.mitre.org/XMLSchema/oval-definitions-5#ios">
46.      <version_string operation="pattern match">12\.3\((\d+\w*\)?[A-Z]{1,2}(\d.*|$)</
       version_string>
47.    </version55_state>
48.    <line_state id="oval:org.mitre.oval:ste:6946" version="1" xmlns="http://
       oval.mitre.org/XMLSchema/oval-definitions-5#ios">
49.      <show_subcommand>show running-config</show_subcommand>
50.      <config_line operation="pattern match">\s+5060($|\s+)</config_line>
51.    </line_state>
52.  </states>

53. </oval_definitions>
```

Listing 2: Cisco IOS vulnerability example (part 2)

System (IOS) [44] in order to overview the OVAL main building blocks. It is based on a real vulnerability specification but it was simplified to show how a basic OVAL definition looks like. An OVAL definition is typically written in one XML file but here we divide it in two parts just for didactic purposes. The first part illustrated in Listing 1 contains the OVAL definition that represents our vulnerability description. A definition is the key structure in OVAL. It is analogous to the logical sentence or proposition: if a computer's state matches the configuration parameters laid out in the criteria, then that computer exhibits the state described. Within this example, the vulnerability definition with id *oval:org.mitre.oval:def:6086* (lines 9-26) states that the referred vulnerability is present on the system if both following conditions hold: (i) the IOS version belongs to a set of affected IOS versions (line 23), and (ii) VoIP is configured (line 24). The second part of our example illustrated in Listing 2 defines the rest of required components referred on the first part, namely, OVAL tests (lines 28-37), OVAL objects (lines 38-43) and OVAL states (lines 44-52). An OVAL Test is used by one or more definitions to compare an object(s) against a defined state. An OVAL Object describes a unique set of items to look for on a system. This unique set of items can then be used by an OVAL Test and compared against an OVAL State. An OVAL State is a collection of one or more characteristics pertaining to a specific object type. The OVAL State is used by an OVAL Test to determine if a unique set of items identified on a system meet certain characteristics. The first condition is analyzed by the first test with id *oval:org.mitre.oval:tst:9025* (lines 29-32). This *version test* refers to one OVAL object (line 39) and one OVAL state (lines 45-47). It will be true if and only if the specified object match the specified state. The *pattern match* expression allows to specify a family of IOS versions using a regular expression (line 46). The second condition is analyzed by the second test with id *oval:org.mitre.oval:tst:24211* (lines 33-36). This *line test* refers to one OVAL object (lines 40-42) and one OVAL state (lines 48-51). It will be true if and only if the sub-command *show running-config* results contains the port number *5060* (line 50).

The OVAL language currently constitutes a de facto standard for describing vulnerabilities as well as good practices. Autonomic environments should take advantage of this capability in order to augment their vulnerability awareness as shown in [45], [46], later explained in Section V-A. In deed, several related languages have evolved around the OVAL language. The National Institute of Standards and Technology (NIST) [7] has supported the development of the Security Content Automation Protocol (SCAP) [14]. The SCAP protocol is a suite of specifications that standardize the format and nomenclature by which security software communicate information about publicly known software flaws and security configurations annotated with common identifiers and embedded in XML. SCAP also utilizes software flaw and security configuration standard reference data, also known as SCAP content. This reference data is provided by the National Vulnerability Database (NVD) [47], which is managed by NIST and supported by the Department of Homeland Security (DHS) [48]. Other public vulnerability databases exist as well such as the Open Source Vulnerability Database (OSVDB) [12], though the vulnerability descriptions provided by them are usually understandable by humans and not by machines, thus difficulting an automated consumption of this security knowledge. SCAP can be used for several purposes, including automating vulnerability checking, technical control compliance activities, and security measurement. The integration of

SCAP into self-governing environments constitutes a major challenge, however its automation-targeted nature can highly benefit future autonomics development. The SCAP protocol includes the OVAL language and complements it with enumeration languages such as the Common Platform Enumeration (CPE), a nomenclature and dictionary of hardware, operating systems, and applications [49]; the Common Configuration Enumeration (CCE), a nomenclature and dictionary of security software configurations [50]; and CVE for enumerating security-related software flaws. SCAP also considers the eXtensible Configuration Checklist Description Format (XCCDF) for authoring security benchmarks and reporting checklist evaluation results [51], and the Common Vulnerability Scoring System (CVSS) for measuring and scoring the relative severity of software flaw vulnerabilities [52]. The specifications involved in the SCAP protocol, particularly OVAL and XCCDF, not only allows us to specify vulnerabilities, but also to bring a system into compliance through the remediation of identified vulnerabilities or misconfigurations. XCCDF is intended to serve as a replacement for the security hardening and analysis documentation written in prose. In other words, XCCDF provides means for describing specific machine states and performing certain actions when these states are present on the system. These features perfectly fit requirements for expressing which actions autonomic systems should take when vulnerable states are detected. CVSS on the other hand, provides a framework for quantifying the impact of vulnerabilities. The normalized score computed for each vulnerability based on several types of metrics provides a strong support for risk assessment techniques and enables the prioritization of actions to take when change management mechanisms are performed. Such metrics can be very useful within the decision-making process required to be implemented by self-governing systems.

These specifications highly contribute to security automation and to the vulnerability management activity, this survey being focused on the assessment of vulnerabilities in autonomic environments. Other works have been done as well such as the one proposed in [53] where an ontology-based approach for dealing with vulnerability management activities called OVM is presented. However, its connection with autonomic technologies is not addressed, nor the scalability or actual impact on real networks. Moreover, OVM only considers the vulnerability management activities from a high-level perspective, focusing on the process rather than fine-grained concepts that allow vulnerabilities to be described. Languages such as OVAL are crucial for representing security knowledge that in turn involves technical details. The OVAL language has been further detailed as a means for performing the assessment activity. Standardization efforts are essential for exchanging this knowledge and it requires a strong support of the community. Autonomic networks and systems should be able to manage these security advisories and capitalize the knowledge provided by vulnerability descriptions repositories in order to increase their vulnerability awareness. Moreover, autonomic elements should be able to provide appropriate sensing and actuation mechanisms as depicted in Fig. 4 in order to be autonomously assessed and eventually corrected. In this section we have investigated different mechanisms for describing vulnerabilities and exchanging related information that provide a strong support for achieving this goal.

V. Detecting Vulnerabilities

Once a vulnerability is known and described, mechanisms used for detecting it constitute a central concern on autonomic networks and systems. Self-governed environments should be able to incorporate and take advantage of security advisories provided by different sources when vulnerability assessment activities are performed. In this section we will discuss different methods and systems for assessing both device and network vulnerabilities contributing to the third dimension of the D^3 approach , and we will present several approaches for correlating security information and analyzing potential attacks and security policies violations.

A. Analyzing device vulnerabilities

The assessment of local vulnerabilities on a device requires the investigation of specific states and conditions that may allow an attacker to compromise the system. While black-box techniques, such as network scanning discussed in subsection V-B, can provide useful security information without requiring specific tools in the device under analysis, grey-box techniques can highly enhance the obtained information by accessing the device itself and inspecting its internal state and particular configurations. Assessing vulnerabilities using the OVAL language can be understood as a grey-box approach since it not only allows to specify vulnerability descriptions but also standardizes the three main steps of the assessment process, namely, representing configuration information of systems for testing; analyzing the system for the presence of the specified machine state (vulnerability, configuration, patch state, etc.); and reporting the results of the assessment. Fig. 7 describes the main steps of the OVAL process [43]. At step 1, security advisories are published and encoded as OVAL definitions at step 2. These definitions are then interpreted at step 3 to gather all the required information in order to perform

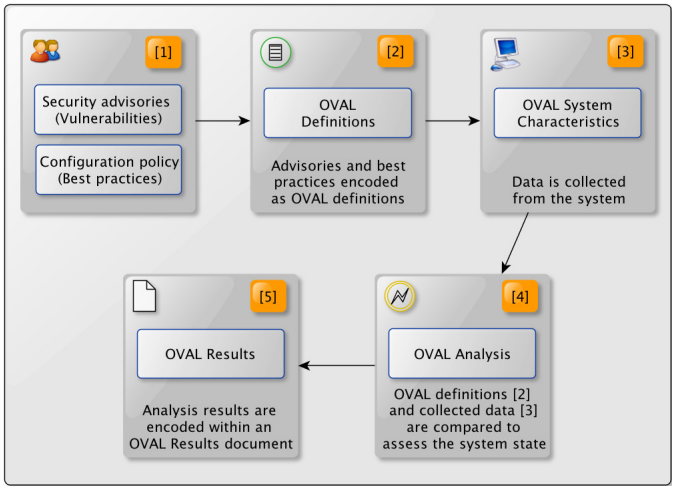

Fig. 7: OVAL-based vulnerability assessment [43]

the analysis at step 4. Once the OVAL analysis is done, a report is generated at step 5 identifying if the specific machine states described at step 2 are present or not on the target system. The integration of such process into the management plane of self-governing environments is feasible as demonstrated in [45] and later explained, providing a strong basis for autonomously assessing the exposure of autonomic elements. It is important to notice that OVAL is a specification language and it allows to describe content; real analysis is performed by OVAL interpreters. However, interpreter's activity is guided by the underlying OVAL language structure, thus we can think of OVAL as a language for specifying, analyzing and reporting vulnerabilities. Moreover, because OVAL allows to describe specific machine states, semantics can be used in several ways, i.e., states that can not hold (vulnerabilities), states that should hold (best practices).

Several OVAL-based systems have been developed since the OVAL language was released. The work proposed in [54] presents the design and implementation of a vulnerability assessment tool based on the OVAL language to detect weak points in Linux System. The proposed approach has more readability, reliability, scalability and simplicity than traditional tools. Although this work was published in 2004, it clearly highlights OVAL's potentiality. Others up-to-date OVAL-based tools exist as well. Ovaldi [55] is a free OVAL interpreter maintained by MITRE intended to provide a reference implementation for evaluating OVAL definitions. Current releases of the interpreter cover a wide, but not complete, part of OVAL's specification. This incomplete coverage arises difficulties to extrapolate its usage within other fields such as forensic scenarios. Although Ovaldi is a robust tool, its main development language (C) is platform-dependent, thus increasing maintenance efforts for each OVAL supported platform. Moreover, its internal design and continuous official releases make it quite difficult to use as a base start point for customized or extended OVAL-based tools. The work proposed in [56] presents XOvaldi, a live forensic, multi-platform and extensible OVAL-based system for digital evidence collection. XOvaldi has been purely written in Java [57] and its plugin-based architecture as well as its automatic model adaptation provide easy means for naturally evolving with dynamic forensics scenarios. In deed, an OVAL-based distributed framework for performing self-assessment activities using an extension to XOvaldi for the Android platform [58] has been proposed in [46]. This work presents a lightweight framework that uses a repository of OVAL definitions for Android as its knowledge source. When new security advisories become available, mobile devices are automatically fed with such knowledge in order to analyze their own exposure. In addition, changes affecting Android components trigger efficient reassessment activities by only evaluating known vulnerabilities that involve those affected components.

The effort invested in the development of OVAL-based assessment systems provides a strong background for automating the detection of known vulnerabilities. These systems can be then combined and integrated into autonomic environments in order to enhance their ability for detecting security threats. Indeed, a novel approach for increasing the vulnerability awareness of self-governed environments has been proposed in [45]. It addresses the integration and assessment of vulnerability descriptions into the management plane of autonomic networks and systems by taking advantage of the security knowledge provided by OVAL repositories and the Cfengine system. Cfengine is an autonomic maintenance system that provides support for automating the management of large-scale environments based on high-level policies [59], [60]. In this work, an OVAL to Cfengine translation tool called Ovalyzer is presented, which is capable of automatically producing Cfengine policy rules that represent OVAL security advisories, thus enabling autonomous agents to assess themselves their own exposure.

Autonomic networks must be capable of adapting according to specific policies or security issues. Because of this, analyzing network vulnerabilities positions strong challenges that autonomic entities should be able to solve. Network vulnerability analysis, also known as vulnerability scanning, involves activities to determine vulnerabilities and security holes exploitable within the target network. In order to perform this analysis, data collection and examination has to be done over members of the network and correlation techniques must be applied to analyze the target network as a whole.

B. Analyzing network vulnerabilities

The ability of identifying host-based and distributed vulnerabilities constitutes the first step for the vulnerability management process to be completely embedded into the management plane of autonomic networks and systems. Network scanning constitutes one of the most used techniques for discovering devices in a network. This process allows to identify active hosts either for security assessment or for performing different kinds of attacks. The enumeration of a network provides useful information such as users, groups and running services on each network member. Port scanners are usually used within this process for analyzing each device in order to detect which ports are open and which services are listening on them. Probes against these ports and the behavior presented by the target device may allow port scanners to infer useful information about the software running on each port such as the type of application and its version.

The kind of response emitted by the device under analysis indicates whether the port is in use, and if so, it can be further explored for detecting weaknesses. Fingerprinting for instance, is a technique used for interpreting the responses of an operating system by sending to it different combinations of data and analyzing its responses against a fingerprint database [61]. Fingerprints are usually generated by the application of a hash function over a specific piece of data where the obtained hash value uniquely identifies the input data. Behavior patterns for well and bad-formed messages are correlated with the observed responses in order to obtain a match of known systems and applications, and related vulnerabilities. Currently, several network scanners exist for assessing vulnerabilities on a target network such as Nessus [62], OpenVAS [63] or SAINT [64]. Some of them use the functionalities provided by powerful port scanners such as Nmap [65]. However, these tools do

not provide standard means for describing and exchanging the vulnerabilities they are able to assess. Languages such as OVAL are highly required. In addition, none of them have currently shown trends or means for being embedded into self-governed environments.

Regardless of the mechanisms used for individually assessing devices, grey-box techniques such as agent-based vulnerability assessment or black box techniques such as network scanning, it is essential to develop approaches capable of analyzing the network and its relations as a whole. Steps taken by an intruder usually respond to some favorable conditions present on the system. By modeling these capabilities and actions to take, inference can be performed. Reasoning engines are widely used in this field to achieve automated approaches. As an example, the work presented in [66] and enhanced in [67], [68] introduces a logic-based network security analyzer called MulVAL. MulVAL is a framework and reasoning system that conducts multi-host, multi-stage vulnerability analysis on a network. Its purpose is to model the interaction of software bugs with system and network configurations. MulVAL uses Datalog as its modeling language, thus the information in the vulnerability database provided by the bug-reporting community, the configuration information of each machine and the network, and other relevant information are all encoded as Datalog facts. The proposed framework uses the OVAL language to analyze each host on the network. The reasoning engine consists of a collection of Datalog rules that captures the operating system behavior and the interaction of various components in the network. After gathering all required information from the environment, the analysis performed by MulVAL has two main parts. First, all possible multi-steps accesses and inferred privileges on each user are computed. Then, results are compared against the stated global policy. If the analysis results show a user with some kind of privilege that is not present on the global policy, a security breach has been found. Due to autonomic networks are by nature governed by policies, such approach and the methodology used are particularly appropriated to be embedded into policy-driven environments such as autonomic networks and systems.

Even though the ability of evaluating devices by looking for vulnerable states as OVAL does highly increases the overall security of networked self-governing environments, there exist scenarios where securing a network by individually assessing its member devices is not enough. While each network device may present a secure state, a combination of them may cause a distributed vulnerable state across the network. The work presented in [69] proposes a mathematical approach for formally defining the concept of a distributed vulnerability as well as a framework for assessing distributed vulnerabilities in autonomic networks. The authors present DOVAL, an OVAL-based language capable of expressing distributed vulnerable scenarios and propose a computable infrastructure for assessing such security advisories over a consolidated view of the network in order to detect vulnerable states that may simultaneously involve two or more devices. Fig. 8 illustrates the assessment process performed over autonomic networks using the DOVAL language. At step 1 security advisories describing distributed vulnerabilities are encoded as DOVAL definitions. These definitions are translated to Cfengine policy rules at step 2 and consumed by a Cfengine server. A minimized loop-free topology of the underlying network is generated at step 3. At step 4 the required information about hosts and the network are gathered and aggregated at the root of the generated spanning tree. DOVAL definitions are assessed over the gathered data at step 5 and a DOVAL report is generated in order to inform the assessment results identifying potential threats on the network under analysis. This work has been extended in [70] where a collaborative remediation framework for distributed vulnerabilities is presented. The proposed approach depicts an XCCDF-based language as well as a distributed framework for performing collaborative vulnerability treatments in autonomics environments.

C. Correlating vulnerabilities with threats and attack graphs

The mechanisms used for detecting vulnerabilities in autonomic networks provide an extremely useful overview of the potential security problems that might be exploited on a target network. However, this information can be yet enhanced by correlating security threats found in the assessment phase as shown in Fig. 4 and analyzing how the activity of an attacker could take advantage of them. Attack modeling languages such as ACML [71] allows to express the capability gained by an attacker at each step of the intrusion process. This approach allows to link network events and detect multi-steps attacks. This can be very useful in the context of autonomic environments as it could support the analysis of scenarios where an autonomic element has been compromised. In that context, not only such element but also those connected to it and the relationships between them, e.g. service provisioning requirements, should be analyzed as well. Such investigation could provide an insightful security overview and the possibilities that an attacker might have taking advantage of vulnerable autonomic elements in the network. Previous work done in [72] considers the idea of a requires-provide model where each gained privilege by an attacker opens new intrusion capabilities. This concept is extremely important

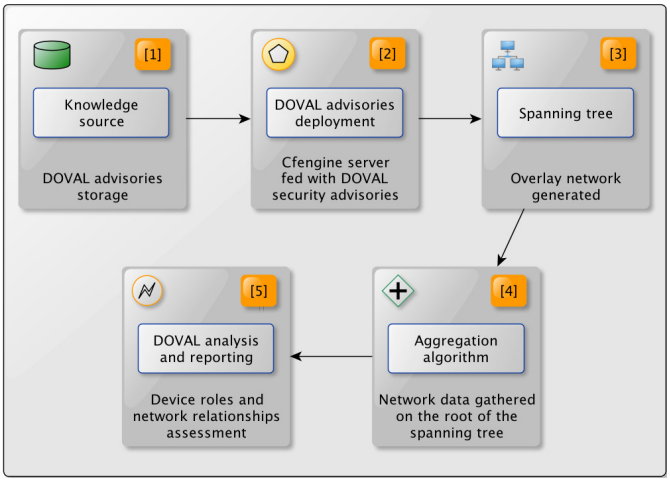

Fig. 8: Distributed vulnerability assessment with DOVAL [69]

when analyzing attack sequences and provides robust foundations for attack graphs approaches. A deep review on attack graphs is presented in [73] where several contributions on this topic are analyzed. The authors make clear the achievements and limitations of attack graphs by discussing fundamental construction concepts as well as their use in network security approaches. The work performed in [74] describes a graph-based network vulnerability analysis approach, revisiting the idea of attack graphs themselves and providing a compact and scalable graph representation. Although the authors show that it is possible to produce attack trees using their representation, they argue that more useful information can be produced, for larger networks, while bypassing the attack tree step. The proposed approach relies on an explicit assumption of monotonicity, which, in essence, states that the precondition of a given exploit is never invalidated by the successful application of another exploit. This means that the attacker never needs to backtrack. The assumption reduces the complexity of the analysis problem from exponential to polynomial, thereby bringing even very large networks within reach of analysis. Considering the growing management complexity that the autonomic paradigm aims at dealing with, such assumption still holds and the ability to understand the interconnections between autonomic elements in a scalable manner provides a promising background for assessing security weaknesses as a whole. Nevertheless, these approaches should be also extended to capture the nature of autonomics. As explained before, autonomic elements interact with the environment by means of sensors and effectors. This perspective differs a bit from the traditional one where active services are usually waiting for clients on known ports. Therefore, this issue should also be considered within approaches aiming at analyzing autonomic entities and their interconnections as well.

It is important to notice the difference between vulnerabilities and attack graphs. While a vulnerability represents a potential security problem that could be exploited by an attacker in order to compromise the system, an attack graph describes the actual activity and steps performed by an attacker in order to achieve a desired goal. In other words, a vulnerability is focused on the system by identifying unsecure states and an attack graph is focused on the behavior of the attacker that takes advantage of these security weaknesses. Recently, the Common Attack Pattern Enumeration and Classification (CAPEC) language [75] has been proposed by MITRE for describing attack patterns. CAPEC involves a collection of common methods for exploiting software systems, including network attack patterns. The CAPEC schema also enables the use of the Cyber Observable eXpression (CybOX) language [76] as a means for describing cyber observables that exist for various steps and portions of the attack pattern. Such cyber observables refer to events or stateful measures that can be observed in the operational domain. Other standardization efforts that use CybOX exist as well such as the Malware Attribute Enumeration and Characterization language (MAEC) [77] for characterizing the behavior of malware and the Common Event Expression (CEE) [78] for unifying the representation and classification of events found in the lifecycle of systems and networks. Currently, MITRE is also considering automated mechanisms for converting MAEC and CybOX content into OVAL checks in order to detect malware artifacts and other host-based cyber observables. These initiatives are still in an early stage though they are quite promising as their contributions might harden the security of autonomic environments.

Network attacks are complex and several approaches from disparate computing fields such as artificial intelligence have been proposed as well. For instance, an expert system has been described in [79] where the system is based on a decision tree that uses predetermined invariant relationships between redundant digital objects to detect semantic incongruities. By searching for violations of known data relationships, an attacker's unauthorized change may be automatically identified providing useful information for deeper analysis. Self-governing environments may benefit of this approach by specifying such relationships in the form of policies that can be then controlled by the underlying autonomic government mechanisms. In order to detect how an attack can be performed on a system, an automated diagnosis model can be built such as the one presented in [80]. This model takes as input the victim configuration scenario, information about existent vulnerabilities, and privileges gained by an attacker; and generates possible attack sequences allowing to identify assumptions about the attacker's behavior. A successful attack sequence simulation indicates a feasible means of accomplishing the unauthorized access. Such approach can be used as both proactive and reactive forensic mechanisms for securing vulnerable autonomic elements. The work proposed in [81] addresses the network vulnerability problem with test cases, which amount to attack scenarios, generated by a model checker. Model checking is a technique for automatically verifying correctness properties of finite-state systems where a given system model can be automatically tested to evaluate whether such model meets a given specification. The authors encode the vulnerabilities in a state machine description suitable for a model checker and then assert that an attacker cannot acquire a given privilege on a given host. The model checker either offers assurance that the assertion is true on the actual network or provides a counter example detailing each step of a successful attack. Even though this work does not ensure scalability features which is a critical factor in autonomics, it is worth exploiting such paradigm when the behavior and properties of the underlying self-governed environment can be formally specified.

While some authors focus on attacks anatomy, other authors also propose metrics for quantifying attack potentiality which depends on several factors. The work presented in [82] proposes a framework for measuring the vulnerability of individual hosts based on current and historical operational data for vulnerabilities and attacks. Most approaches have examined software flaws only, not other vulnerabilities such as software misconfigurations and software feature misuse. The framework uses a highly automatable metrics-based approach, producing rapid and consistent measurements for quantitative risk assessment and for attack and vulnerability modeling. Metrics are particularly important within autonomic environments as they can be used for autonomously parametrizing the behavior of the entire system. Such measurements can be successfully integrated in the closed loop that depicts the lifecycle of self-

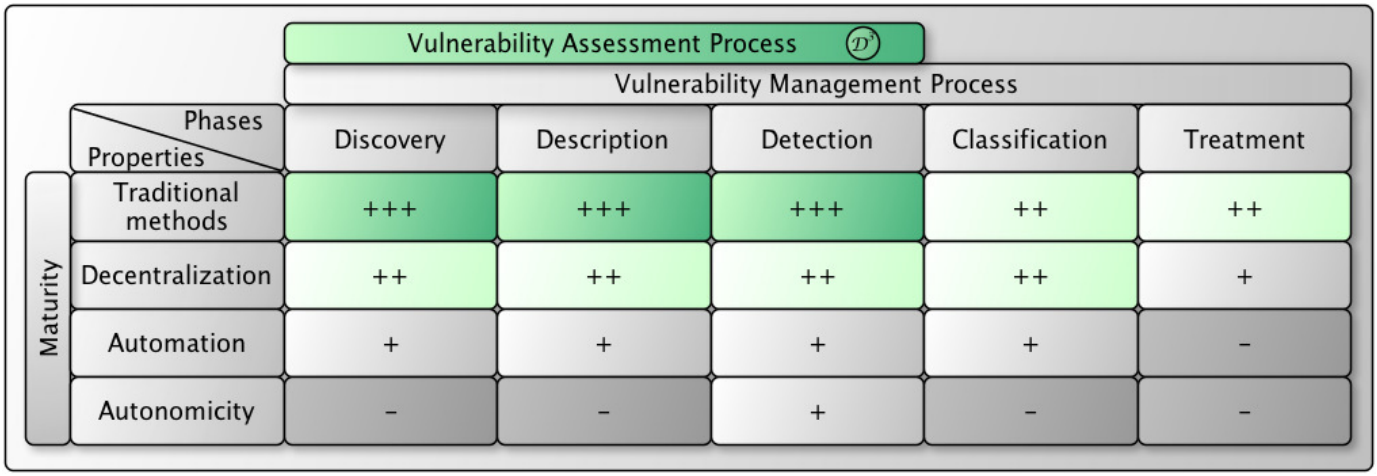

Fig. 9: Scientific maturity of vulnerability management activities with respect to autonomic networks

governed elements in order to feed and control their behavior. An equally interesting method is presented in [83] where a policy security metric is calculated based on a number of factors like vulnerabilities present in the system, vulnerability history of the services and their exposure to the network, and traffic patterns. The proposed approach combines computed scores into one unified score called *Policy Security Score*. This score can be later used to judge how good a policy is, as well as compare policies and assess policy changes. Such approach provides support not only for assessing the dynamics of individual policy-based self-governed systems but also for evaluating the overall behavior of autonomic environments in which vulnerabilities play a critical role.

Assessing vulnerabilities constitutes a crucial activity that enables autonomic networks and systems to identify threats that potentially may compromise their security. This ability is in turn complemented by approaches capable of correlating exploitable network weaknesses in order to identify potential successful attacks. The integration of such mechanisms within the management plane of autonomic environments provides a powerful basis for assessing their own exposure. In this section we have presented methods for detecting device and network vulnerabilities, and we have discussed different approaches for correlating security information and inferring potential attacks.

VI. Future research challenges

During the realization of our investigation, we have detected several challenges that must be addressed in order to be able to really integrate the autonomic computing approach into daily computer systems and networks. In that context, Fig. 9 summarizes the scientific maturity of the vulnerability management process with respect to autonomic environments highlighting properties and issues that should be further investigated. As depicted in the first column, traditional methods for discovering unknown vulnerabilities count with a strong foundation, though decentralized as well as automated approaches require further investigation. Autonomic methods for addressing this capability have been barely or even at all discussed. Autonomic computing should incorporate these capabilities in order to unveil potential existing threats. As to describing known vulnerabilities, shown in the second column, several scientific contributions have been done, mostly from a device perspective. However, automatic generation as well as autonomic mechanisms for describing security problems are still in an early stage thus requiring research efforts in order to harden the foundations and maturity of such activity. Autonomic environments should capitalize such security knowledge in order to analyze themselves and assess their own exposure. A variety of methods have been proposed for detecting vulnerabilities in non-autonomic environments as shown in the third column of Fig. 9. However, decentralized mechanisms exist in a minor degree, and automated and autonomic approaches have been weakly discussed. Once security problems are detected, they need to be classified according to their impact and risk, and remediated through the application of appropriate treatments. Vulnerability classification and treatment mechanisms, fourth and fifth columns of Fig. 9 respectively, have only been partially addressed in the past for non-autonomic environments and mostly from a centralized perspective. Automated and autonomic approaches for dealing with these activities remain an open problem.

In light of this, we have observed several lacks mostly located on the automation and autonomicity properties of Fig. 9 that should be further investigated. We highlight here three transversal research axes that are important to leverage the maturity and robustness of these properties:

- Integration of vulnerability models into the management plane of networks and services. This integration requires automated means for exchanging vulnerability descriptions in a standardized manner as well as detecting them. Vulnerability detection can be performed by dynamically translating vulnerability descriptions into configuration policy rules interpretable by autonomic configuration systems. In addition, such perspective can be enriched with automated vulnerability discovery mechanisms. This feature can enable the alignment of network components to desired and secure states. However, mechanisms for dealing with rules conflicts and policy consistency must be in place as well.
- Investigation of collaborative methods and techniques

for performing vulnerability management activities in a decentralized manner, with multiple vulnerability description datasources. Autonomic elements need automated mechanisms for healing security holes. Control mechanisms and algorithms for classifying vulnerabilities and executing vulnerability treatments (apriori or aposteriori configuration operations) in an optimal manner must be analyzed. The SCAP protocol and particularly, the XCCDF language, can be extremely useful to achieve this point.

- Formalized approaches for supporting the two previous themes are highly required. Robust data collection mechanisms, mature system assessment techniques and efficient environment discovery methods constitute cornerstones within the integration of the vulnerability management process in self-governed environments. Autonomic networks and systems can take advantage of disparate computing fields such as digital forensics for threat discovery, machine learning for adaptation, or statistical models for prediction. Methods for managing and planning changes as well as techniques for assessing their impact are essential within the vulnerability management process. Reasoning systems capable of capitalizing security knowledge can provide new horizons for dealing with dynamic environments, not only from an operational viewpoint but from a security perspective too.

VII. Conclusions

The continuous growth of networks as well as the diversification of their services have considerably increased the complexity of their management. In order to face this problem, the autonomic computing paradigm aims at defining a strong basis for automated systems capable of managing themselves in an autonomous manner considering properties such as self-configuration, self-protection, self-optimization and self-healing. However, when administration tasks and self-management activities are performed, changes may lead to vulnerable states increasing the exposure of the environment to security threats, thus it is essential to count on change management mechanisms and risk assessment techniques. In this scenario, vulnerability management activities are crucial for ensuring the safety of such systems and hence, their functionalities and stability. In this survey we have explored in depth different vulnerability assessment challenges that are critical for completely integrating the vulnerability management process into such autonomic environments.

As systems and technologies evolve, new space for vulnerabilities comes into scene. Autonomic networks and systems must be able to perform auto-analysis and detect potential security problems that may be exploited by malicious entities, thus the vulnerability assessment activity constitutes a major challenge. In this survey we have proposed a novel perspective called the D^3 approach that classifies the vulnerability assessment activity into three well defined dimensions: *Discovery, Description* and *Detection*. The D^3 approach provides a framework where different problems and potential solutions concerning the integration of vulnerability assessment activities into the management plane of autonomic environments can be analyzed in an organized manner. In that context, background and key concepts as well as different leading methods and current techniques have been discussed along this work. We have identified potential applications over diverse contributions that may provide a strong basis for achieving this critical goal within self-governing systems. In addition, we also have pointed out several areas such as vulnerability integration models, collaborative vulnerability management approaches and policy-based reasoning systems where the development of novel approaches and solutions are required to provide autonomic environments with the ability of assessing their own exposure.

As happens in the real world, autonomic elements coexist within dynamic environments, interacting with others autonomic and non-autonomic elements. Nevertheless, there are also continuous threats that may compromise autonomic elements safety. If an autonomic element is compromised, its functions and abilities become untrustworthy and eventually disabled; thus autonomic elements that use services of the former become compromised as well. This inevitably leads to distrust and failure of the autonomic environment. Autonomic systems must be able to manage their own state and perform required activities to achieve secure configurations. Autonomic elements unable to support this capability will age with time, becoming more vulnerable, insecure and useless. It is the belief of the authors that automation is possible only if autonomic networks and systems fully integrate vulnerability management mechanisms for ensuring safe configurations.

Acknowledgements

This work was partially supported by the EU FP7 Univerself Project, FI-WARE PPP and the Flamingo Project.

References

[1] J. Kephart and D. Chess, "The Vision of Autonomic Computing," *Computer*, vol. 36, pp. 41–50, Jan. 2003.

[2] S. Dobson, F. Zambonelli, S. Denazis, A. Fernández, D. Gaïti, E. Gelenbe, F. Massacci, P. Nixon, F. Saffre, and N. Schmidt, "A Survey of Autonomic Communications," *ACM Transactions on Autonomous and Adaptive Systems*, vol. 1, pp. 223–259, Dec. 2006.

[3] N. Samaan and A. Karmouch, "Towards Autonomic Network Management: an Analysis of Current and Future Research Directions," *IEEE Communications Surveys & Tutorials*, vol. 11, pp. 22–36, July 2009.

[4] M. C. Huebscher and J. A. McCann, "A Survey of Autonomic Computing–Degrees, Models, and Applications," *ACM Comput. Surv.*, vol. 40, pp. 7:1–7:28, August 2008.

[5] Z. Movahedi, M. Ayari, R. Langar, and G. Pujolle, "A Survey of Autonomic Network Architectures and Evaluation Criteria," *IEEE Communications Surveys & Tutorials*, vol. PP, pp. 1–27, May 2011.

[6] IBM, "An Architectural Blueprint for Autonomic Computing," *IBM White Paper*, 2006.

[7] "NIST, National Institute of Standards and Technology." http://www.nist.gov/. Last visited on January, 2013.

[8] V. Igure and R. Williams, "Taxonomies of Attacks and Vulnerabilities in Computer Systems," *IEEE Communications Surveys & Tutorials*, vol. 10, pp. 6–19, Jan. 2008.

[9] O. Dabbebi, R. Badonnel, and O. Festor, "Dynamic Exposure Control in P2PSIP Networks," in *NOMS*, pp. 261–268, 2012.

[10] P. Foreman, *Vulnerability Management*. Information Security, CRC Press, 2009.

[11] R. E. Ball, *The Fundamentals of Aircraft Combat Survivability Analysis and Design, 2nd Edition*. AIAA Education Series, 2003.

[12] "OSVDB, The Open Source Vulnerability Database." http://osvdb.org/. Last visited on January, 2013.

[13] "CVE, Common Vulnerabilities and Exposures." http://cve.mitre.org/. Last visited on January, 2013.

[14] J. Banghart and C. Johnson, "The Technical Specification for the Security Content Automation Protocol (SCAP). Nist Special Publication." http://scap.nist.gov/revision/, 2011. Last visited on January, 2013.

[15] "IBM Autonomic Computing Deployment Model." http://www-03.ibm.com/press/us/en/pressrelease/464.wss.

[16] A. Williams and M. Nicolett, "Improve IT Security with Vulnerability Management." http://www.gartner.com/id=480703, 2005. Last visited on January, 2013.

[17] "ITSM - IT Service Management." http://www.itsm.info/. Last visited on January, 2013.

[18] Y. Diao, A. Keller, S. Parekh, and V. V. Marinov, "Predicting Labor Cost through IT Management Complexity Metrics," *In Proceedings of the 10th IFIP/IEEE International Symposium on Integrated Network Management (IM'07)*, pp. 274–283, May 2007.

[19] A. Tang, A. Nicholson, Y. Jin, and J. Han, "Using Bayesian Belief Networks for Change Impact Analysis in Architecture Design," *Journal of Systems and Software*, vol. 80, pp. 127–148, Jan. 2007.

[20] J. Sauve, R. Santos, R. Reboucas, A. Moura, and C. Bartolini, "Change Priority Determination in IT Service Management Based on Risk Exposure," *IEEE Transactions on Network and Service Management*, vol. 5, pp. 178–187, Sept. 2008.

[21] J. A. Wickboldt, L. A. Bianchin, and R. C. Lunardi, "Improving IT Change Management Processes with Automated Risk Assessment," *In Proceedings of the IEEE International Workshop on Distributed Systems: Operations and Management (DSOM'09)*, pp. 71–84, 2009.

[22] T. Setzer, K. Bhattacharya, and H. Ludwig, "Decision Support for Service Transition Management - Enforce Change Scheduling by Performing Change Risk and Business Impact Analysis," *In Proceedings of the IEEE Network Operations and Management Symposium (NOMS'08)*, pp. 200–207, Apr. 2008.

[23] R. Costa Cardoso and M. M. Freire, "Towards Autonomic Minimization of Security Vulnerabilities Exploitation in Hybrid Network Environments," *In Proceedings of the Joint International Conference on Autonomic and Autonomous Systems and International Conference on Networking and Services (ICAS-ISNS'05)*, 2005.

[24] M. S. Ahmed, E. Al-Shaer, M. M. Taibah, M. Abedin, and L. Khan, "Towards Autonomic Risk-aware Security Configuration," *Proceedings of the IEEE Network Operations and Management Symposium (NOMS'08)*, pp. 722–725, Apr. 2008.

[25] M. S. Ahmed, E. Al-Shaer, and L. Khan, "A Novel Quantitative Approach For Measuring Network Security," in *Proceedings of the 27th IEEE Conference on Computer Communications (INFOCOM'08)*, pp. 1957 –1965, April 2008.

[26] R. Bohme, "Vulnerability Markets. What is the Economic Value of a Zero-Day Exploit?," in *Proceedings of the 22nd Chaos Communication Congress*, December 2005.

[27] S. Frei, D. Schatzmann, B. Plattner, and B. Trammel, "Modelling the Security Ecosystem - The Dynamics of (In)Security," in *Proceedings of the Workshop on the Economics of Information Security (WEIS'09)*, June 2009.

[28] R. Patton, *Software Testing (2nd Edition)*. Sams, 2005.

[29] J. Demott, "The Evolving Art of Fuzzing. Software Testing." http://vdalabs.com/tools/The_Evolving_Art_of_Fuzzing.pdf, 2006. Last visited on January, 2013.

[30] T. Wang, T. Wei, G. Gu, and W. Zou, "TaintScope: A Checksum-Aware Directed Fuzzing Tool for Automatic Software Vulnerability Detection," in *Proceedings of the IEEE Symposium on Security and Privacy (SP'10)*, pp. 497 –512, May 2010.

[31] H. Dai, C. Murphy, and G. Kaiser, "Configuration Fuzzing for Software Vulnerability Detection," *2010 International Conference on Availability, Reliability and Security*, pp. 525–530, Feb. 2010.

[32] V. Corey, C. Peterman, S. Shearin, M. Greenberg, and J. Van Bokkelen, "Network Forensics Analysis," *Internet Computing, IEEE*, vol. 6, pp. 60 – 66, Nov 2002.

[33] "A Road Map for Digital Forensic Research," in *Report From the First Digital Forensic Research Workshop (DFRWS)*, (New York, NY, USA), August 2001.

[34] H. Achi, A. Hellany, and M. Nagrial, "Network Security Approach for Digital Forensics Analysis," *In Proceedings of the International Conference on Computer Engineering and Systems (CCES'08)*, pp. 263–267, Nov. 2008.

[35] W. Wang and T. E. Daniels, "A Graph-based Approach Toward Network Forensics Analysis," *ACM Transactions on Information and System Security (TISSEC)*, vol. 12, no. 1, 2008.

[36] M. J. Khan, M. M. Awais, and S. Shamail, "Enabling Self-Configuration in Autonomic Systems Using Case-Based Reasoning with Improved Efficiency," *In Proceedings of the 4th International Conference on Autonomic and Autonomous Systems (ICAS'08)*, pp. 112–117, Mar. 2008.

[37] H. M. Tran and J. Schönwälder, "Distributed Case-Based Reasoning for Fault Management," in *Proceedings of the 1st international conference on Autonomous Infrastructure, Management and Security: Inter-Domain Management (AIMS'07)*, (Berlin, Heidelberg), pp. 200–203, Springer-Verlag, 2007.

[38] "MITRE Corporation." http://www.mitre.org/. Last visited on January, 2013.

[39] J. Caballero, Z. Liang, P. Poosankam, and D. Song, "Towards Generating High Coverage Vulnerability-Based Signatures with Protocol-Level Constraint-Guided Exploration," in *Proceedings of the 12th International Symposium on Recent Advances in Intrusion Detection (RAID'09)*, pp. 161–181, Springer-Verlag, 2009.

[40] "RFC 4765." http://www.ietf.org/rfc/rfc4765.txt. Last visited on January, 2013.

[41] "Vulnerability Naming Schemas and Description Languages: CVE, Bugtraq, AVDL and VulnXML. The SANS Institute." http://www.sans.org/. Last visited on January, 2013.

[42] "AVDL." http://www.oasis-open.org/. Last visited on January, 2013.

[43] "The OVAL Language." http://oval.mitre.org/. Last visited on January, 2013.

[44] "Cisco IOS." http://www.cisco.com/. Last visited on January, 2013.

[45] M. Barrère, R. Badonnel, and O. Festor, "Supporting Vulnerability Awareness in Autonomic Networks and Systems with OVAL," *In Proceedings of the 7th IEEE International Conference on Network and Service Management (CNSM'11)*, Oct. 2011.

[46] M. Barrère, G. Hurel, R. Badonnel, and O. Festor, "Increasing Android Security using a Lightweight OVAL-based Vulnerability Assessment Framework," *In Proceedings of the 5th IEEE Symposium on Configuration Analytics and Automation (SafeConfig'12)*, Oct. 2012.

[47] "NVD, National Vulnerability Database." http://nvd.nist.gov/. Last visited on January, 2013.

[48] "DHS, Department of Homeland Security." http://www.dhs.gov/. Last visited on January, 2013.

[49] "CPE, Common Platform Enumeration." http://cpe.mitre.org/. Last visited on January, 2013.

[50] "CCE, Common Configuration Enumeration." http://cce.mitre.org/. Last visited on January, 2013.

[51] N. Ziring and S. D. Quinn, "Specification for the Extensible Configuration Checklist Description Format (XCCDF). NIST (National Institute of Standards and Technology)." http://scap.nist.gov/specifications/xccdf/. Last visited on January, 2013.

[52] "CVSS, Common Vulnerability Scoring System." http://www.first.org/cvss/. Last visited on January, 2013.

[53] J. A. Wang and M. Guo, "OVM: An Ontology for Vulnerability Management," in *Proceedings of the 5th Annual Workshop on Cyber Security and Information Intelligence Research: Cyber Security and Information Intelligence Challenges and Strategies (CSIIRW'09)*, (New York, NY, USA), pp. 34:1–34:4, ACM, 2009.

[54] Y. Kwon, H. J. Lee, and G. Lee, "A Vulnerability Assessment Tool Based on OVAL in Linux System," *Network and Parallel Computing*, pp. 653–660, 2004.

[55] "Ovaldi, the OVAL Interpreter reference implementation." http://oval.mitre.org/language/interpreter.html. Last visited on January, 2013.

[56] M. Barrère, G. Betarte, and M. Rodríguez, "Towards Machine-assisted Formal Procedures for the Collection of Digital Evidence," in *Proceedings of the 9th Annual International Conference on Privacy, Security and Trust (PST'11)*, pp. 32 –35, July 2011.

[57] "Java technology." http://www.oracle.com/technetwork/java/. Last visited on January, 2013.

[58] "Android." http://www.android.com/. Last visited on January, 2013.

[59] "Cfengine." http://www.cfengine.com/. Last visited on January, 2013.

[60] M. Burgess and Æ. Frisch, *A System Engineer's Guide to Host Configuration and Maintenance Using Cfengine*, vol. 16 of *Short Topics in System Administration*. USENIX Association, 2007.

[61] G. F. Lyon, *Nmap Network Scanning: The Official Nmap Project Guide to Network Discovery and Security Scanning*. USA: Insecure, 2009.

[62] "Nessus." http://www.tenable.com/products/nessus. Last visited on January, 2013.

[63] "OpenVAS." http://www.openvas.org/. Last visited on January, 2013.

[64] "Saint." http://www.saintcorporation.com/. Last visited on January, 2013.

[65] "Nmap." http://nmap.org/. Last visited on January, 2013.

[66] X. Ou, S. Govindavajhala, and A. W. Appel, "MulVAL: A Logic-based Network Security Analyzer," *on USENIX Security*, 2005.

[67] X. Ou, W. F. Boyer, and M. A. McQueen, "A Scalable Approach to Attack Graph Generation," in *Proceedings of the 13th ACM Conference on Computer and Communications Security (CCS'06)*, pp. 336–345, ACM Press, 2006.

[68] D. Saha, "Extending Logical Attack Graphs for Efficient Vulnerability Analysis," in *Proceedings of the 15th ACM Conference on Computer and Communications Security (CCS'08)*, (New York, NY, USA), pp. 63–74, ACM, 2008.

[69] M. Barrère, R. Badonnel, and O. Festor, "Towards the Assessment of Distributed Vulnerabilities in Autonomic Networks and Systems," *In Proceedings of the IEEE/IFIP Network Operations and Management Symposium (NOMS'12)*, Apr. 2012.

[70] M. Barrère, R. Badonnel, and O. Festor, "Collaborative Remediation of Configuration Vulnerabilities in Autonomic Networks and Systems," *In Proceedings of the 8th IEEE International Conference on Network and Service Management (CNSM'12)*, Oct. 2012.

[71] N. K. Pandey, S. K. Gupta, S. Leekha, and J. Zhou, "ACML: Capability Based Attack Modeling Language," *In Proceedings of the 4th International Conference on Information Assurance and Security*, pp. 147–154, Sept. 2008.

[72] S. J. Templeton and K. Levitt, "A Requires/Provides Model for Computer Attacks," *In Proceedings of the Workshop on New Security Paradigms (NSPW'00)*, pp. 31–38, 2000.

[73] R. Lippmann, K. Ingols, and L. Laboratory, *An Annotated Review of Past Papers on Attack Graphs*. Project report IA, Massachusetts Institute of Technology, Lincoln Laboratory, 2005.

[74] P. Ammann, D. Wijesekera, and S. Kaushik, "Scalable, Graph-based Network Vulnerability Analysis," *In Proceedings of the 9th ACM Conference on Computer and Communications Security (CCS'02)*, p. 217, 2002.

[75] "CAPEC, Common Attack Pattern Enumeration and Classification." http://capec.mitre.org/. Last visited on January, 2013.

[76] "CybOX, Cyber Observable eXpression." http://cybox.mitre.org/. Last visited on January, 2013.

[77] "MAEC, Malware Attribute Enumeration and Characterization." http://maec.mitre.org/. Last visited on January, 2013.

[78] "CEE, Common Event Expression." http://cee.mitre.org/. Last visited on January, 2013.

[79] T. Stallard and K. Levitt, "Automated Analysis for Digital Forensic Science: Semantic Integrity Checking," *In Proceedings of the 19th Annual Computer Security Applications Conference (ACSAC'03)*, pp. 160–167, 2003.

[80] C. Elsaesser and M. C. Tanner, "Automated Diagnosis for Computer Forensics," tech. rep., The Mitre Corporation, 2001.

[81] R. W. Ritchey and P. Ammann, "Using Model Checking to Analyze Network Vulnerabilities," *In Proceedings of the IEEE Symposium on Security and Privacy (SP'00)*, pp. 156–165, 2000.

[82] K. Scarfone and T. Grance, "A Framework for Measuring the Vulnerability of Hosts," *In Proceedings of the 1st International Conference on Information Technology (ICIT'08)*, pp. 1–4, May 2008.

[83] M. Abedin, S. Nessa, E. Al-Shaer, and L. Khan, "Vulnerability Analysis for Evaluating Quality of Protection of Security Policies," *In Proceedings of the 2nd ACM Workshop on Quality of Protection (QoP'06)*, 2006.

Martín Barrère is a Ph.D. Candidate at the University of Lorraine working in the MADYNES Research Team at INRIA, France. He has received his computer engineering degree in 2010 by the University of the Republic of Uruguay. His current research topic is vulnerability management in autonomic networks and systems, which includes network security, autonomic computing, logic and formal languages, and configuration management. He is also interested in digital forensics, particularly on evidence collection methods and techniques.

Rémi Badonnel is an Associate Professor at TELECOM Nancy and a research staff member of the MADYNES Research Team at INRIA. Previously he worked on change management methods and algorithms at IBM T.J. Watson in USA and on autonomous systems at the University College of Oslo in Norway. His research interests include network and service management, dynamic and autonomic environments, security and defence techniques.

Prof. Olivier Festor is the Director of TELECOM Nancy and head of the MADYNES Research Team at INRIA. His research interests are in the design of algorithms and solutions for automated management of large-scale networks and services. Chair of the IFIP TC6 WG6.6 and Co-Chair of the IRTF NMRG, he serves in several TPCs as well as in the editorial boards of the major international conferences and journals in Network and Service Management.